特别提示：本代码为所有顶点的数据类型由整型改为双精度后代码，这样可避免因类型转换引起精度损失。其中被替代的代码仍保留，但已被注释掉。

特别提示：本代码为所有顶点的数据类型由整型改为双精度后代码，这样可避免因类型转换引起精度损失。其中被替代的代码仍保留，但已被注释掉。

功能：节点细化。对包括均匀、准均匀、分段贝齐尔、一般非均匀和包绕型B样条闭曲线的节点矢量批量插入节点。
输入参数：m_manyR为重复插入相异节点的次数序列；
　　　　　m_manyM为插入节点序列在原节点矢量中已有的重复度序列；
　　　　　m_manyU为插入递增的相异节点序列；
　　　　　未列入形参的控制顶点(m_xnewVertex,m_ynewVertex)、节点矢量m_newNode、次数m_nTimes，均为被保护成员。
输出参数：p、q分别为插入的相异节点序列在原节点矢量中左右端节点下标；插入节点序列后生成的新控制顶点(含未改变的原顶点) (m_xnewVertex,m_ynewVertex)
　　　　　与节点矢量m_newNode均为被保护成员。InsertTimes-总的插入次数，即增加控制顶点数，公有成员。
调用函数： GetKnotsRepeats为获得顶点数为n+1的k次B样条曲线的节点矢量TemNode中的相异节点数组Knots与重复度数组(两者均为公有成员)。

```
void InsertManyU(CArray<int,int> &m_manyR,CArray<int,int> &m_manyM,CArray<double,double> &m_manyU,int &p,int &q)
{
        int j,pj,ii,jj;
        int k=m_nTimes;
//获得要被插入的节点区间左端节点下标值
        for(ii=0;ii<m_newNode.GetSize();ii++)
                if(m_manyU[0]<m_newNode[ii]) {p=ii-1;   break;}
//获得要被插入的节点区间右端节点下标值
        for(ii=0;ii<m_newNode.GetSize();ii++)
                if(m_manyU[m_manyU.GetSize()-1]<m_newNode[ii]) {q=ii;   break;}
//确定总的插入次数，即增加的顶点数InsertTimes
        for(ii=0;ii<m_manyU.GetSize();ii++)
        {
                int Rii=m_manyR[ii];
                InsertTimes=InsertTimes+Rii;
        }
//确定节点矢量m_newNode中的相异节点数组Knots与重复度数组Repeats及其长度l。
//      int n=m_newVertex.GetSize()-1;
        int n=m_xnewVertex.GetSize()-1;
        CArray<double,double> TemNode;
        TemNode.SetSize(m_aNode.GetSize());
        for(ii=0;ii<m_aNode.GetSize();ii++) TemNode[ii]=m_newNode[ii];
        GetKnotsRepeats(k,n,TemNode);
//开始按节点值递增顺序插入节点过程
        for(j=0;j<m_manyU.GetSize();j++)
        {
                double aU=m_manyU[j];  //插入的相异节点值
                int l=m_manyR[j];      //重复插入相异节点值aU的次数
                int r=m_manyM[j];      //该相异节点值aU在原节点矢量中已有的重复度
//确定节点矢量m_newNode中的相异节点数组Knots与重复度数组Repeats及其长度l。
//              int n=m_newVertex.GetSize()-1;
                int n=m_xnewVertex.GetSize()-1;
//              CArray<CPoint,CPoint> m_temp1;
                CArray<double,double> m_xtemp1,m_ytemp1;
//确定第j个待插相异节点m_manyU[j]在m_manyR[j-1]次插入m_manyU[j-1]生成的节点矢量中所在的节点区间左端节点下标pj
                for(ii=1;ii<Knots.GetSize();ii++)
                        if(aU>=Knots[ii-1]&&aU<Knots[ii]) break;
                int pp=0;
                for(int i3=0;i3<ii;i3++) pp=pp+Repeats[i3];
                pj=pp-1;
//给出第j个待插相异节点m_manyU[j]涉及的老控制顶点m_tempVertex。这里将原始顶点称为原顶点，相对于本回插入生成的新顶点，将上回插入生成的新顶点称为老顶点。
//              m_tempVertex.RemoveAll();
                m_xtempVertex.RemoveAll();  m_ytempVertex.RemoveAll();
//将m_manyR[j]次插入第j个待插相异节点m_manyU[j]称为第j回插入。插入后,如果m_manyU[j]=0.，则节点数与顶点数保持不变；否则，顶点数增加l=m_manyR[j]个。

if((m_OpenClose==1&&aU==0.)||(m_OpenClose==0&&(aU==m_newNode[k]||aU==m_newNode[m_newNode.GetSize()-k-1])))
                {
//                      for(jj=pj-k;jj<=pj-r;jj++) m_temp1.Add(m_newVertex[jj]);
                        for(jj=pj-k;jj<=pj-r;jj++) {m_xtemp1.Add(m_xnewVertex[jj]); m_ytemp1.Add(m_ynewVertex[jj]);}
//                      if(l+r<k) m_tempVertex.SetSize(k-r+l-1);
                        if(l+r<k) {m_xtempVertex.SetSize(k-r+l-1);  m_ytempVertex.SetSize(k-r+l-1);}
//                      if(l+r==k) m_tempVertex.SetSize(2*l-1);
                        if(l+r==k) {m_xtempVertex.SetSize(2*l-1);  m_ytempVertex.SetSize(2*l-1);}
//                      if(l+r==k+1) m_tempVertex.SetSize(2*(l-1)-1);
```

```
                        if(l+r==k+1) {m_xtempVertex.SetSize(2*(l-1)-1);
m_ytempVertex.SetSize(2*(l-1)-1);}
                        for(int s=1;s<=l;s++)
                        {
                                for(jj=pj-k;jj<=pj-r-s;jj++)
                                {
                                        int h=pj-k;
                                        double alpha;
                                        if((m_newNode[jj+k+1]-m_newNode[jj+s])==0.) alpha=0.;
                                        else
alpha=(aU-m_newNode[jj+s])/(m_newNode[jj+k+1]-m_newNode[jj+s]);
//
m_temp1[jj-h].x=int((1-alpha)*m_temp1[jj-h].x+alpha*m_temp1[jj-h+1].x);
//
m_temp1[jj-h].y=int((1-alpha)*m_temp1[jj-h].y+alpha*m_temp1[jj-h+1].y);
                                        m_xtemp1[jj-h]=(1-alpha)*m_xtemp1[jj-h]+alpha*m_xtemp1[jj-h+1];
                                        m_ytemp1[jj-h]=(1-alpha)*m_ytemp1[jj-h]+alpha*m_ytemp1[jj-h+1];
                                }
                                if(m_OpenClose==0&&r+l==k+1&&(aU==0.||aU==1.))
                                {
                                        if(aU==0.)
                                        {
//                                              if(s<l) m_tempVertex[s-1]=m_temp1[0];
                                                if(s<l) {m_xtempVertex[s-1]=m_xtemp1[0];
m_ytempVertex[s-1]=m_ytemp1[0];}
//                                              if(s==l) m_tempVertex[s-2]=m_temp1[0];
                                                if(s==l) {m_xtempVertex[s-2]=m_xtemp1[0];
m_ytempVertex[s-2]=m_ytemp1[0];}
//                                              for(jj=1;jj<=k-r-1;jj++)
m_tempVertex[jj+l-2]=m_temp1[jj];
                                                for(jj=1;jj<=k-r-1;jj++)
{m_xtempVertex[jj+l-2]=m_xtemp1[jj];  m_ytempVertex[jj+l-2]=m_ytemp1[jj];}
                                        }
//                                      if(aU==1.&&s<l) m_tempVertex[s-1]=m_temp1[0];
                                        if(aU==1.&&s<l) {m_xtempVertex[s-1]=m_xtemp1[0];
m_ytempVertex[s-1]=m_ytemp1[0];}
                                }
                                if(r+l<=k)
                                {
//                                      m_tempVertex[s-1]=m_temp1[0];
                                        m_xtempVertex[s-1]=m_xtemp1[0];   m_ytempVertex[s-1]=m_ytemp1[0];
//                                      for(jj=1;jj<=k-r-1;jj++) m_tempVertex[jj+l-1]=m_temp1[jj];
                                        for(jj=1;jj<=k-r-1;jj++) {m_xtempVertex[jj+l-1]=m_xtemp1[jj];
m_ytempVertex[jj+l-1]=m_ytemp1[jj];}
                                }
                        }
                        if(m_OpenClose==1&&aU==0.)
                        {
//                              for(jj=pj-k;jj<=pj-r;jj++) m_newVertex[jj-pj+k]=m_temp1[jj-pj+k];
                                for(jj=pj-k;jj<=pj-r;jj++) {m_xnewVertex[jj-pj+k]=m_xtemp1[jj-pj+k];
m_ynewVertex[jj-pj+k]=m_ytemp1[jj-pj+k];}
//                              for(jj=0;jj<pj-r;jj++) m_newVertex[n+1-pj+r+jj]=m_tempVertex[jj];
                                for(jj=0;jj<pj-r;jj++) {m_xnewVertex[n+1-pj+r+jj]=m_xtempVertex[jj];
m_ynewVertex[n+1-pj+r+jj]=m_ytempVertex[jj];}
                                int newNodeCount=m_newNode.GetSize();
                                for(jj=0;jj<=k-l-1;jj++) m_newNode[jj]=m_newNode[jj+l];
                                for(jj=newNodeCount-1;jj>=newNodeCount-k+l;jj--)
m_newNode[jj]=m_newNode[jj-l];
                                for(jj=1;jj<=l;jj++)
                                {
                                        m_newNode[k-jj]=m_newNode[k];
                                        m_newNode[newNodeCount-1-k+jj]=m_newNode[newNodeCount-1-k];
                                }
                        }
                        if(m_OpenClose==0&&aU==m_newNode[k])
                        {
//                              for(jj=1;jj<=k-1;jj++)
m_newVertex[jj-1]=m_tempVertex[m_tempVertex.GetSize()-k+jj];
                                for(jj=1;jj<=k-1;jj++)
{m_xnewVertex[jj-1]=m_xtempVertex[m_xtempVertex.GetSize()-k+jj];
m_ynewVertex[jj-1]=m_ytempVertex[m_ytempVertex.GetSize()-k+jj];}
//                              m_tempVertex.SetSize(k-1);
                                m_xtempVertex.SetSize(k-1);  m_ytempVertex.SetSize(k-1);
//                              for(jj=0;jj<k-1;jj++) m_tempVertex[jj]=m_newVertex[jj];
                                for(jj=0;jj<k-1;jj++) {m_xtempVertex[jj]=m_xnewVertex[jj];
m_ytempVertex[jj]=m_ynewVertex[jj];}
```

```
                                for(jj=0;jj<=k-r-l;jj++) m_newNode[jj]=m_newNode[l+jj];
                                for(jj=k-r-l+1;jj<=k-1;jj++) m_newNode[jj]=m_newNode[k];

                        }
                        if(m_OpenClose==0&&aU==m_newNode[m_newNode.GetSize()-k-1])
                        {
//                              for(jj=1;jj<=k-1;jj++)
m_newVertex[m_newVertex.GetSize()-k+jj]=m_tempVertex[jj-1];
                                for(jj=1;jj<=k-1;jj++)
{m_xnewVertex[m_xnewVertex.GetSize()-k+jj]=m_xtempVertex[jj-1];
m_ynewVertex[m_ynewVertex.GetSize()-k+jj]=m_ytempVertex[jj-1];}
//                              m_tempVertex.SetSize(k-1);
                                m_xtempVertex.SetSize(k-1);  m_ytempVertex.SetSize(k-1);
                                int n=m_newNode.GetSize()-1;
                                for(jj=n;jj>=n-k+r+1;jj--) m_newNode[jj]=m_newNode[jj-1];
                                for(jj=n-k+r;jj<=n-k+r+l-1;jj++) m_newNode[jj]=m_newNode[n-k];
                        }
                }
                if(aU!=0.&&aU!=1.)
                {
                        int r0=1;  //r0为定义域首端节点的重复度
                        for(ii=k-1;ii>=0;ii--) if(m_newNode[ii]==m_newNode[k]) r0++;
//                      m_newVertex.SetSize(n+l+1);
                        m_xnewVertex.SetSize(n+l+1);  m_ynewVertex.SetSize(n+l+1);
                        //l=m_manyR[j]次插入第j个相异节点aU=m_manyU[j]生成k-r+l-1个新顶点，这里
r=m_manyM[j]是aU在原节点矢量中已有的重复度。
//                      m_tempVertex.SetSize(k-r+l-1);
                        m_xtempVertex.SetSize(k-r+l-1);  m_ytempVertex.SetSize(k-r+l-1);
                        //l=m_manyR[j]次插入第j个相异节点aU=m_manyU[j]涉及k-r+1个老顶点。
//                      for(ii=pj-k;ii<=pj-r;ii++) m_temp1.Add(m_newVertex[ii]);
                        for(ii=pj-k;ii<=pj-r;ii++) {m_xtemp1.Add(m_xnewVertex[ii]);
m_ytemp1.Add(m_ynewVertex[ii]);}
                        //将m_newVertex中第pj-r个及后面的顶点下标后移1,腾出因插入节点增加顶点所要求的空
间。
//                      for(ii=n;ii>=pj-r;ii--) m_newVertex[ii+l]=m_newVertex[ii];
                        for(ii=n;ii>=pj-r;ii--) {m_xnewVertex[ii+l]=m_xnewVertex[ii];
m_ynewVertex[ii+l]=m_ynewVertex[ii];}
                        //插入第j个待插相异节点m_manyU[j]，进行l=m_manyR[j]次。
                        for(int s=1;s<=l;s++)
                        {
                                for(jj=pj-k;jj<=pj-r-s;jj++)                   //教材第7章式(7.6b)德布尔
算法递推公式
                                {
                                        int h=pj-k;
                                        double alpha;
//                                      alpha=(aU-m_newNode[jj+s])/(m_newNode[jj+k+1]-m_newNode[jj+s]);
                                        alpha=(aU-m_newNode[jj+s])/(m_newNode[jj+k+1]-m_newNode[jj+s]);
//
m_temp1[jj-h].x=int((1-alpha)*m_temp1[jj-h].x+alpha*m_temp1[jj-h+1].x);
//
m_temp1[jj-h].y=int((1-alpha)*m_temp1[jj-h].y+alpha*m_temp1[jj-h+1].y);
                                        m_xtemp1[jj-h]=(1-alpha)*m_xtemp1[jj-h]+alpha*m_xtemp1[jj-h+1];
                                        m_ytemp1[jj-h]=(1-alpha)*m_ytemp1[jj-h]+alpha*m_ytemp1[jj-h+1];
//                                      m_tempVertex[s-1]=m_temp1[0];
                                        m_xtempVertex[s-1]=m_xtemp1[0];  m_ytempVertex[s-1]=m_ytemp1[0];
                                }
//                              for(jj=1;jj<=k-r-1;jj++) m_tempVertex[jj+l-1]=m_temp1[jj];
                                for(jj=1;jj<=k-r-1;jj++) {m_xtempVertex[jj+l-1]=m_xtemp1[jj];
m_ytempVertex[jj+l-1]=m_ytemp1[jj];}
                        }
//                      for(jj=pj-k+1;jj<=pj-r+l-1;jj++) m_newVertex[jj]=m_tempVertex[jj-pj+k-1];
                        for(jj=pj-k+1;jj<=pj-r+l-1;jj++)
                        {
                                m_xnewVertex[jj]=m_xtempVertex[jj-pj+k-1];
m_ynewVertex[jj]=m_ytempVertex[jj-pj+k-1];
                                double xjj=m_xnewVertex[jj];   //测试
                                double yjj=m_ynewVertex[jj];   //测试
                        }
                        if(m_OpenClose==1&&r0<k)
                        {
                                if(pj<2*k-r0)
//                                      for(jj=0;jj<=k-r0-1;jj++)
m_newVertex[n+l-k+r0+1+jj]=m_newVertex[jj];  //由k-r0+1个顶点m_newVertex[j],j=0,1,...,k定义的首段曲线因插
入节点导致顶点改变,
                                                                                                       //
其中前k-r0个顶点即是末段曲线的后k个顶点，相应改变。
```

```
                                        for(jj=0;jj<=k-r0-1;jj++)
{m_xnewVertex[n+l-k+r0+1+jj]=m_xnewVertex[jj];  m_ynewVertex[n+l-k+r0+1+jj]=m_ynewVertex[jj];} //由k-r0+1
个顶点m_newVertex[j],j=0,1,...,k定义的首段曲线因插入节点导致顶点改变,

                                               //其中前k-r0个顶点即是末段曲线的后k个顶点，相应改变。
                                if(pj>n-k+r0)
//                                      for(jj=0;jj<=k-r0-1;jj++)
m_newVertex[j]=m_newVertex[n+l-k+r0+1+j];    //由k-r0+1个顶点m_newVertex[j],j=0,1,...,k定义的末段曲线因插
入节点导致顶点改变,
                                        for(jj=0;jj<=k-r0-1;jj++)
{m_xnewVertex[j]=m_xnewVertex[n+l-k+r0+1+j];  m_ynewVertex[j]=m_ynewVertex[n+l-k+r0+1+j];}  //由k-r0+1个
顶点m_newVertex[j],j=0,1,...,k定义的末段曲线因插入节点导致顶点改变,

                                            //其中后k-r0个顶点即是首段曲线的前k个顶点，相应改变。
                                                             //其中后k-r0个顶点即是首段曲线的前k个顶点，
相应改变。
                        }
                        CArray<double,double> TemNode;
                        n=m_newNode.GetSize()-1;
                        TemNode.SetSize(n+1);
                        for(jj=0;jj<=n;jj++) TemNode[jj]=m_newNode[jj];
                        if(m_OpenClose==0)
                        {
                                m_newNode.RemoveAll();
                                m_newNode.SetSize(n+l+1);
                                for(jj=0;jj<=pj;jj++) m_newNode[jj]=TemNode[jj];
                                for(jj=n+l;jj>pj+l;jj--) m_newNode[jj]=TemNode[jj-l];
                                for(jj=pj+1;jj<=pj+l;jj++) m_newNode[jj]=aU;
                                int nn=m_newNode.GetSize()-k-2;   //当前控制顶点数-1
                                GetKnotsRepeats(k,nn,m_newNode);
                        }
                        if(m_OpenClose==1)
                        {
                                if(pj>=k&&aU!=TemNode[k]&&pj<=n-k&&aU!=TemNode[n-k])  //若
m_newNode[k]<u<m_newNode[k+1],定义域首端节点重复度r0=r;,若u>=m_newNode[k+1],定义域首端节点重复度r0<r。
                                {
                                        for(jj=0;jj<=n;jj++) TemNode[jj]=m_newNode[jj];
                                        m_newNode.RemoveAll();
                                        m_newNode.SetSize(n+l+1);
                                        for(jj=0;jj<=pj;jj++) m_newNode[jj]=TemNode[jj];
                                        for(jj=n+l;jj>pj+l;jj--) m_newNode[jj]=TemNode[jj-l];
                                        for(jj=pj+1;jj<=pj+l;jj++) m_newNode[jj]=aU;
                                        if(pj>=k&&pj<=2*k&&aU!=TemNode[k])          //定义域首端点后
k+1-r0-l个区间因l次插入同一节点导致定义域末端点后k+1-r0-l个节点与区间发生相应改变
                                        {
                                                for(jj=0;jj<=k;jj++)
m_newNode[n+l-k+jj]=m_newNode[jj+k-r0+1]+1.;
                                        }
                                        if(pj>=n-2*k&&pj<n-k&&aU!=TemNode[n-k])       //定义域末端点前
k+1-r0-l个区间因l次插入同一节点导致定义域首端点前k+1-r0-l个节点与区间发生相应改变
                                        {
                                                for(jj=0;jj<=k;jj++)
m_newNode[jj]=m_newNode[n+l+r0-2*k-1+jj]-1.;
                                        }
                                }
                        }
                }
        }
}
```